Accelerating Exact Logic Synthesis Using Parallel Computing Methods

Austen Ott ❖ Benjamin Zimmer ❖ Dr. Beth Ernst ❖ Dr. Daniel Ernst
Department of Computer Science ❖ University of Wisconsin-Eau Claire

Exact Synthesis

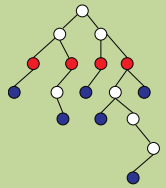
- *Exact synthesis* is the process of creating a minimal-cost representation of a logic function. These representations are used in the design of microprocessors and other digital devices.
- Cost is measured in terms of the number of component parts required to represent the function. These include logic gates and the connections between them.
- Determining the minimal-cost representation is difficult because it involves searching a huge number of possible representations.
- Currently, the most efficient exact synthesis software uses a branch-and-bound algorithm, which lends itself to parallelization. *Parallelization* is the process of dividing the problem into pieces which can be solved simultaneously on multiple processors working cooperatively.

MPI and Clusters

- *MPI* (Message Passing Interface) is a software library that allows programs to send information between any connected processors or processor cores.
- *Clusters* consist of several computers linked together with the intent of combining their processing power. For our research, we used EBWilson, a cluster graciously provided by Dr. Christine Morales of the Chemistry Department.

Our Implementation

- We began with a non-parallel exact synthesis algorithm written by Dr. Beth Ernst. Using the OpenMPI library, we created a working parallel version which demonstrates significant speedup over the original.
- In our implementation, one processor acts like a manager which divides the problem into hundreds of pieces and distributes them to other processors in the cluster. When a processor finishes its task, it returns its results to the manager which then gives it more work, if the manager has pieces left to distribute.
- The pieces of work are best understood as parts of a tree structure. Completing each piece may generate more pieces of work. This tree of work must be fully explored to find a minimal cost solution. In the figure to the right, red nodes represent pieces of work initially given to separate processors, and blue nodes represent terminal points which may be solutions.



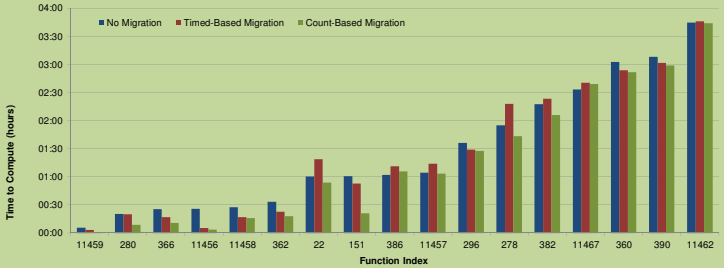
- Once the manager runs out of pieces, it can request work from busy processors to give to processors without work. This way, the program makes the most efficient use of time by keeping all processors continually working. This is called *process migration*.

Versions

- Our implementation can be configured to use one of two different methods of process migration: time-based and count-based.
- *Time-based migration* waits for a specific amount of time after a processor has finished all work before asking other processors to share work.
- *Count-based migration* waits for a specific number of processors to be finished before asking other processors to share work.

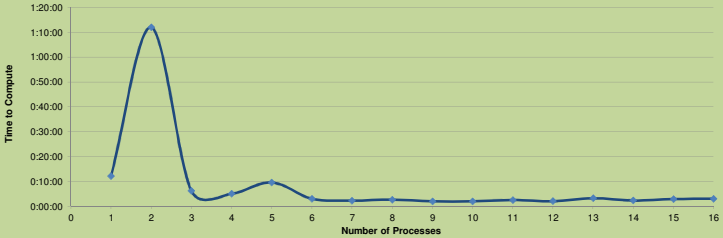
Time Comparisons

Time to Compute vs. Function Index – 16 Processes



Time to Compute vs. Number of Processes for Function 11459

Time-based migration at 0.5 seconds



Future Work

- Optimizing our parallel version will require determining the best values for several parameters:
  - Number of initial divisions of the problem by the manager
  - How long to wait before requesting work from a busy processor
- We plan to modify the program to allow multiple functions to be solved at the same time.
- Finally, we want to use Kraken, a supercomputer, to compute minimal-cost representations of all 5-input logic functions. This will require our program scaling to a much larger number of processors efficiently.

The printing of this poster was funded by Differential Tuition.